VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Systém pro řízení údržby technologických celků pomocí metodiky Condition based Maintenance

System for Maintenance Management of Technological Units using the Condition Based Maintenance

Marek Janouch

Bakalářská práce

Vedoucí práce: Ing. Jan Janoušek

Ostrava, 2021

Abstrakt

Tato bakalářská práce se zabývá tvorbou distribuovaného systému pro řízení údržby pomocí metodiky Condition Based Maintenance. Jednotlivé záznamy o údržbách k zařízením jsou ukládány s využitím skutečně odpracovaných hodin. Kromě plánování systém nabízí správu dat uživatelských a provozních, základní reporty s výstupem do Excelu nebo grafu. V prvních kapitolách jsou popsány základy metodiky CbM a jednotlivé technologie použité k vývoji systému. Dále následuje stručná analýza, návrh celkové aplikace a popis vlastní implementace.

Klíčová slova

Údržba; CbM; PI Systém; C#; MS SQL

Abstract

This thesis deals with the creation of a distributed maintenance management system using the Condition Based Maintenance methodology. All maintenance records for the devices are stored using the real worked hours. In addition to planning, the system offers user and operational data management, basic reports with output to Excel or chart. The first chapters describe the basics of the CbM methodology and the individual technologies used to develop the system. Then there is a brief analysis, design of the overall application and description of the actual implementation.

Keywords

maintenance; CbM; PI System; C#; MS SQL

Poděkování

Rád bych poděkoval Ing. Janu Janouškovi za cenné rady a vstřícnost při konzultacích k této bakalářské práci.

Obsah

Seznam použitých zkratek a symbolů

API	–	Aplication Programming Interface - rozhraní pro programování aplikací
CbM	–	Condition based Maintenance
DBMS	–	Database Management System - systém pro správu databáze
IIoT	–	Industrial Internet of Things - průmyslový internet věcí
IoT	–	Internet of Things - internet věcí
MS	–	Microsoft
RDBMS	–	Relation Database Management System - systém pro správu relační databáze
RPC	–	Remote Procedure Call - vzdálené volání procedur
SQL	–	Structured Query Language - strukturovaný dotazovací jazyk
WCF	–	Windows Communication Foundation
WPF	–	Windows Presentation Foundation
XAML	–	eXtensible Application Markup Language - rozšiřitelný značkovací jazyk pro tvorbu aplikací
XML	–	eXtensible Markup Language - rozšiřitelný značkovací jazyk

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Dnešní doba se dá specifikovat mnoha význačnými milníky, jeden z nich je prudký nárůst moderních technologií. Tomuto se nevyhnul ani průmysl. Zavádění nových procesů, které využívají posledních technologických poznatků a vymožeností, se stalo nejenom moderní, ale také nezbytné k tomu, aby jakýkoliv podnik obstál v těžkém konkurenčním světě. Ve světě Průmyslu 4.0 již nejde obstát jen s tužkou a papírem. Využití výpočetní síly je dnes patrné jak z výroby samotné, ale i podpůrných procesů nezbytných k fungování společnosti.

Jedním z takovýchto procesů je i plánování údržby, které přešlo z modelu "opravíme co se pokazilo" na mnohem efektivnější "udržujme, aby se nepokazilo". Jinak řečeno přechází se z údržby reaktivní na prediktivní. Jelikož je ve většině případů mnohem levnější předpovědět závadu a vyhnout se zastavení výroby, než vyčkávat dokud se něco nepokazí a až následně problém řešit.

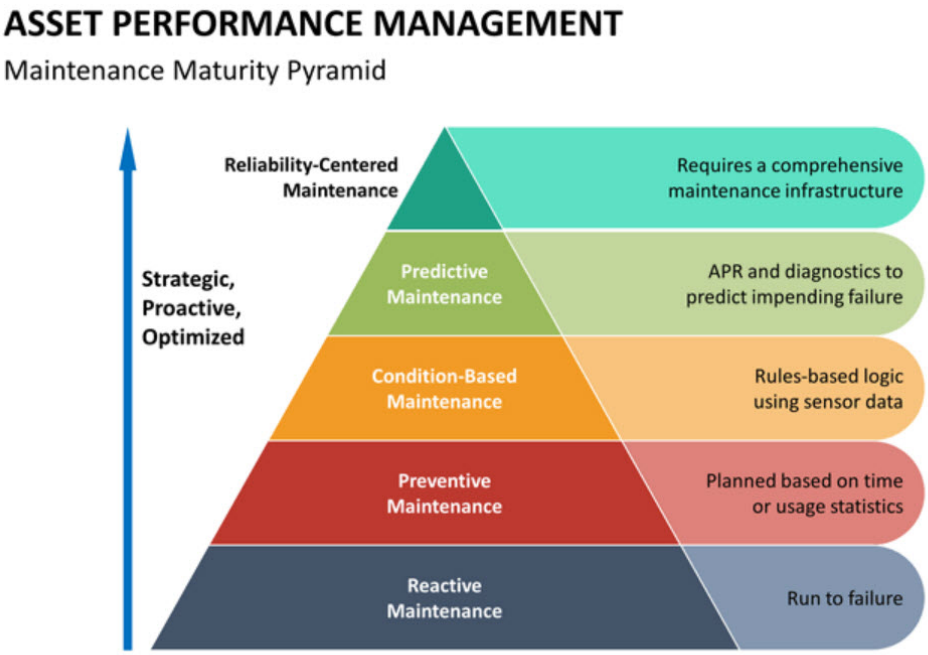
Obsahem této bakalářské práce bude podrobnější pohled na tuto problematiku. Nabídnout vysvětlení základních pojmů a technik týkajících se počítačem řízené údržby. Ukázat stručný seznam firem, které nám mohou v této problematice být nápomocny, jelikož nabízejí poměrně ucelené řešení v podobě odladěných nástrojů, chytrých senzorů, ověřených postupů. Jelikož tyto již hotové systémy nepatří k nejlevnějším a pořizovací náklady mohou často překročit možnosti hlavně menších společností, budu se v závěrečných kapitolách věnovat popisu vlastní implementace. Hlavním kritériem je, aby aplikace umožnila na základě skutečně odpracovaných hodin hlídat stav zařízení, plánovat údržbu a získávat jednoduché a přehledné reporty. Postupně budou popsané technologie použité pro vývoj, dále následuje analýza, návrh samotné aplikace a práce bude zakončena popisem testování.

Kapitola 2

CbM - Condition based Maintenance

Ve výrobních podnicích je údržba klíčovým procesem. Na ni závisí jak produktivita tak i kvalita výroby, což přímo ovlivňuje náklady a zisk společnosti. Jejím cílem je zachování požadované spolehlivosti, dostupnosti a efektivnosti zařízení. A to zejména snaha o prodloužení životnosti zařízení, snížení počtu poruch, zvýšení provozní bezpečnosti či optimalizaci procesů. S rychlým vývojem technologií, nástupem Průmyslu 4.0 či průmyslového internetu věcí (IIoT) dochází ke změně způsobů, jakým se údržba provádí. K dispozici jsou senzory s vysokou přesností, bezdrátové sítě, digitální monitorování stavu zařízení odesílané do cloudů, výkonné softwarové analýzy atd. Dnes se v průmyslu nejčastěji skloňuje tzv. **Pyramida údržby**. Skládá se z 5 stupňů údržby a reprezentuje cestu k proaktivnější a optimalizovanější údržbě.

Základnu pyramidy tvoří **reaktivní údržba**. Jde o nejzákladnější přístup, kdy k údržbě dochází až po zjištění závady nebo po selhání zařízení. Vhodná je pro nekritické zařízení s minimálními náklady na opravu či výměnu. Dalším stupněm je **preventivní údržba**, která se snaží snížit pravděpodobnost poruchy zařízení pravidelnými kontrolami ve fixních časových intervalech nebo na základě zkušeností, statistik a doporučeních výrobce. Uprostřed pyramidy je **Condition based Maintenance**, což můžeme volně přeložit jako údržba na základě stavu zařízení. Tento přístup je vhodný pokud je z provozních dat možné detekovat vznikající poruchu. Sledováním těchto dat spolu s předem definovanými pravidly se automaticky vytvoří pracovní plán, který vyvolá potřebnou údržbu. Čtvrtým stupněm je **prediktivní údržba**, která vyžaduje finančně náročnější, komplexnější a robustnější systém. Kromě provozních dat z různých senzorů a předem definovaných pravidel z CbM je potřebný model výkonu a chování zařízení. Pak se pomocí pokročilých technik analytického modelování detekují anomálie v chování zařízení a predikuje se pravděpodobnost vzniku poruchy v daném časovém intervalu. Na vrcholu pyramidy je **reliability centered maintenance**, která reprezentuje implementaci komplexní optimální strategie údržby na podnikové úrovni. Využívá existující data, pokročilé analýzy, simulace a předpovědi k určení vhodného typu údržby pro každé zařízení. Vhodně nastavená strategie umožňuje vykonávat správné činnosti na správných zařízeních v správném čase, čímž se snižují celkové náklady a zvyšuje se spolehlivost.



Obrázek 2.1: Pyramida údržby [1]

2.1 Metodika CbM

Condition based Maintenance je často zaměňována s preventivní nebo prediktivní údržbou nebo bývá definovaná jako součást jedné z nich. Názory, kam přesně CbM v rámci hierarchie údržby zařadit, se můžou v různých zdrojích lišit. Z pohledu pyramidy údržby na obrázku 2.1 je CbM považována za jakousi zlatou střední cestu mezi preventivní údržbou, která je vykonávaná v pevných předem definovaných časových intervalech (den, týden, měsíc, ...), a prediktivní údržbou, která kombinuje naměřená data se složitými prediktivními vzorci, čímž předpovídá kdy dojde k selhání zařízení.

CbM využívá naměřená data ze senzorů v reálném čase k určení kdy a případně jakou údržbu je potřeba provést. Reaguje tedy na aktuální stav zařízení a pokud sbírané hodnoty vykazují známky nesrovnalostí nebo blížícího se selhání zařízení, provede předem definované akce mající za úkol zabránit poškození zařízení. Za základní myšlenku CbM se považuje provádění údržby pouze v případě potřeby v reakci na skutečný stav zařízení.

Nejčastěji se pomocí senzorů sledují vibrace, rychlost, teplota, tlak, proud, hladiny v nádrži apod. Pro tuto bakalářskou práci bude stěžejní sběr dat otáček motorů nebo jejich proudové odběry na základě kterých se určuje skutečné množství času, kdy bylo zařízení v provozu, tzv. motohodiny.

Aby CbM fungovala správně a byly plně využity její možnosti je potřebné splnit několik požadavků. Dle [2] byl definován standard, jakým by CbM systém měl být technicky navrhnut. Ten

obsahuje 7 základních vrstev:

- **Senzory** - poskytování/sběr dat ze zařízení.
- **Zpracování signálu** - příjem signálu ze senzorů.
- **Monitorování stavu** - porovnání dat s očekávanými hodnotami, generování výstrahy/upozornění.
- **Posouzení kondice** - kontrola zhoršení stavu zařízení, generování diagnostických záznamů, navrhování možné příčiny závady.
- **Prognostika** - odhad budoucího stavu zařízení na základě dat z předešlých vrstev, hlášení budoucího stavu nebo zbývající životnost.
- **Podpora při rozhodování** - generování doporučených akcí a jejich alternativ.
- **Prezentace** - zobrazování důležitých dat z předešlých vrstev.

Každý typ údržby má svoje výhody a nevýhody a CbM není žádnou výjimkou. Vzhledem k tomu, že se údržba provádí pouze když je potřeba, tak jedním z největších přínosů je snížení nákladů a doby strávené údržbou, čímž se zvyšuje spolehlivost a životnost zařízení i jejich výrobní produkce. Tento typ údržby minimalizuje počet neplánovaných prostojů v důsledku selhání zařízení, čímž se snižují i požadavky na nákup náhradních dílů. Plánováním činností optimalizuje intervaly údržby a umožňuje vykonávat opravy mimo provoz zařízení, tj. v době plánovaných odstávek. Velkou výhodou CbM je fungovaní za běhu zařízení, což nezasahuje do běžného provozu.

Nevýhodou tohoto typu údržby je relativně vysoká počáteční investice, která je závislá na aktuálním stavu v podniku - senzory, databáze, existenci jiných systémů, ... Potřeba kvalifikovaného odborníka na analýzu dat, vykonávání práce a rozhodování vyžaduje náklady na zaškolení zaměstnanců. Překážkou k zavedení CbM může být provozní prostředí (např. poškození senzorů vlivem tepla, vlhkosti nebo prachu). Mezi poslední úskalí patří nepředvídatelná doba údržby, což může způsobit, že více zařízení vyžaduje údržbu současně.

2.2 Systémy pro plánování a řízení údržby

Zavedení CbM do podniku je komplexní proces. Je nutné správně vybrat a nainstalovat senzory, zabezpečit přenos a uchovávání velkého množství dat, definovat pravidla údržby, naimplementovat řídící sytém a v neposlední řadě zaškolit zaměstnance pro práci se systémem, aby se korektně rozhodovali a reagovali na výzvy systému.

Na začátku je potřebné analyzovat situaci v podniku, určit, které z nutných podmínek pro provoz systému již jsou k dispozici a co je nezbytné pořídit (např. funguje přenos a uchovávání dat ze senzorů, ale neexistuje systém, který by je zpracovával).

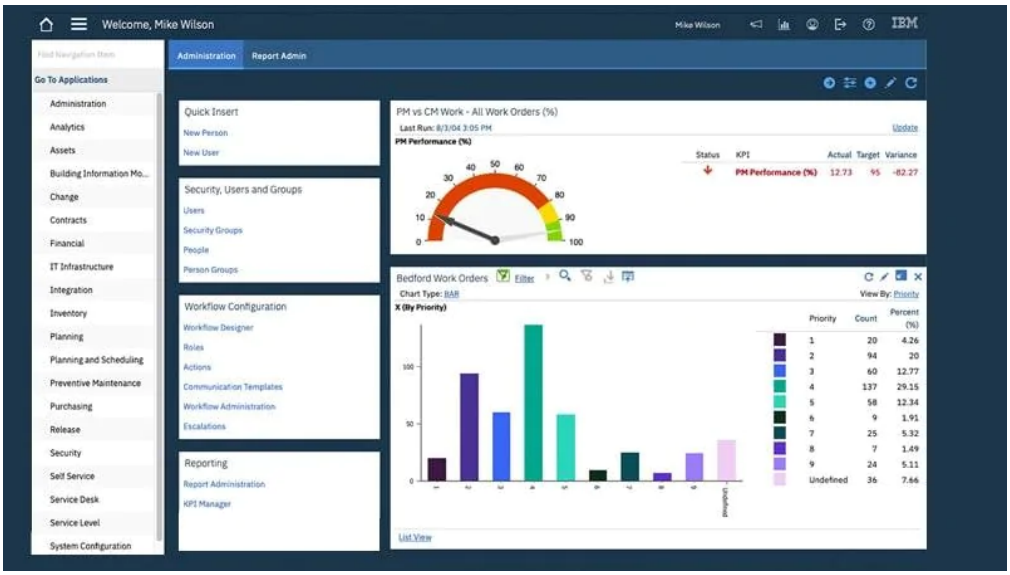
Na trhu je mnoho firem, které se zaobírají danou problematikou a nabízejí částečné nebo úplné řešení. Může se jednat o implementaci systému dle požadavků zákazníka, modulární systémy, kde si zákazník vybere pouze ty části, které potřebuje a napojí je na svůj stávající systém a nebo tzv. krabicové řešení, kde zákazník dostane kompletní předem připravené řešení. Ty mohou obsahovat nejen systém pro plánování a řízení údržby, ale také i zařízení v podobě inteligentních senzorů, možnosti uchovávání provozních dat v lokální či cloudové databázi, nástroje pro vizualizaci dat až po prediktivní údržbu doplněnou například o strojové učení.

Vlivem individuálního přístupu je velmi náročné tyhle nabídky porovnat a následně vybrat vhodné řešení. Rozdíly jsou nejen v technickém řešení, použitých technologiích, uživatelské přívětivosti, licenčních podmínkách, následné technické podpoře ale i ve finanční náročnosti. Ceny řešení se můžou lišit i o desítky milionů korun.

Jako příklady řešení můžeme uvést systém Maximo od firmy IBM, systém COMOS od firmy SIEMENS či moneo od firmy ifm.

Maximo (IBM)

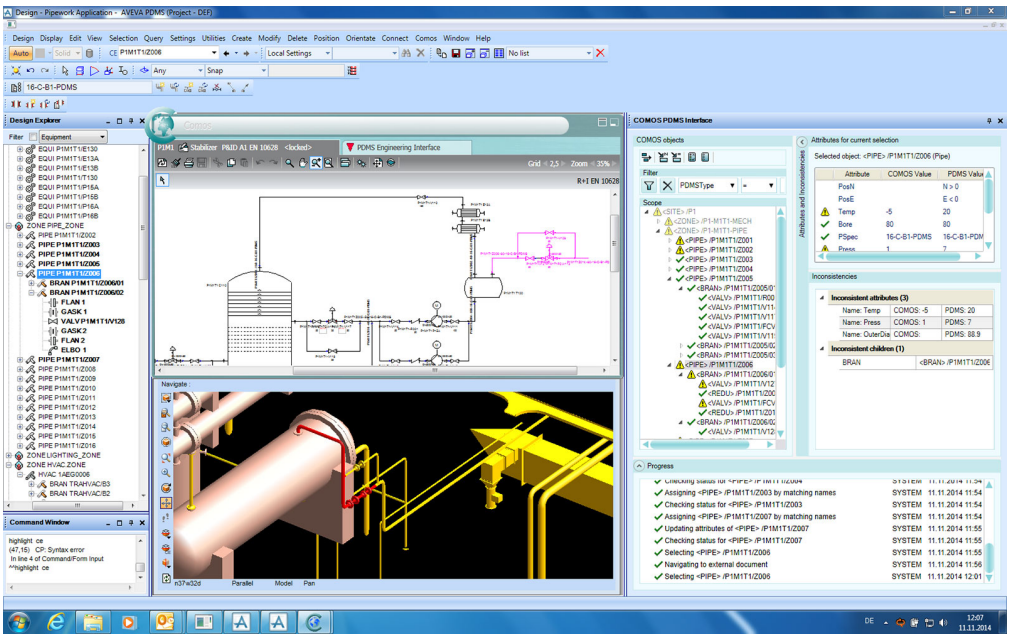
Maximo poskytuje v jedné platformě vzdálené monitorování zařízení, aplikace pro údržbu a spolehlivost, která umožňuje zvýšit provozuschopnost a produktivitu, snížit náklady na údržbu. Mezi klíčové vlastnosti patří integrovaná sada aplikací, zjednodušené licencování, nasazení ve více cloudech a komplexní pohled na zařízení. Jedná se o plně dostupnou a škálovatelnou sadu aplikací, kde se platí pouze za ty aplikace, které se skutečně používají. Rovněž obsahuje správu zařízení, vzdálené monitorování, plánování údržby, bezpečnost pracovníků a mobilní produkty pro správu zařízení.



Obrázek 2.2: IBM asset management [3]

COMOS (SIEMENS)

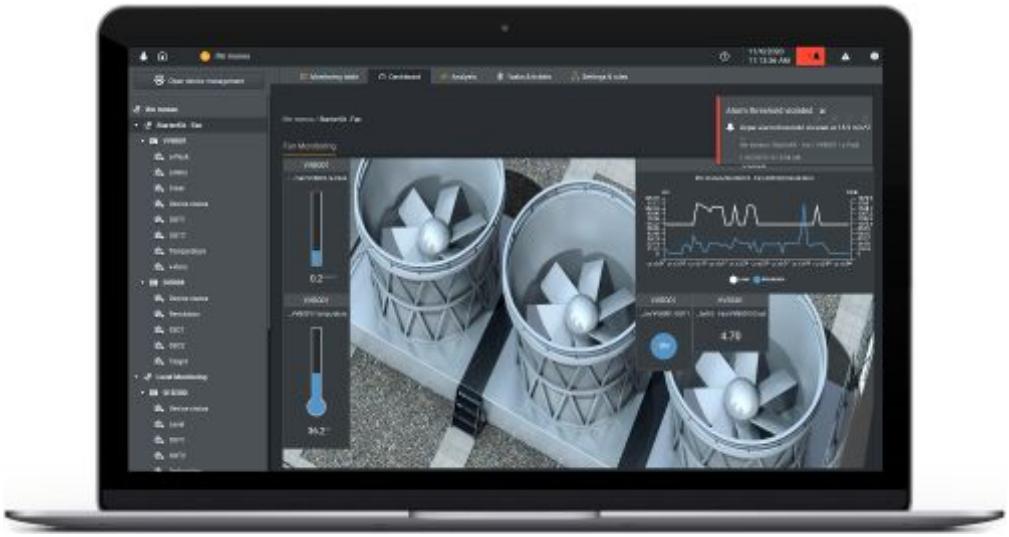
SIEMENS nabízí sofistikované softwarové řešení COMOS pro celistvý management projektu výrobních zařízení od plánování přes provoz a modernizaci až po demontáž. Využitím jediné databáze nedochází ke ztrátě dat v rámci toku informací, aktuální data jsou vždy k dispozici na všech podnikových úrovních a zobrazují stav technologického zařízení v reálném čase. Vysoká modularita umožňuje rychlé a snadné přizpůsobení individuálním potřebám, optimalizovat mezioborovou spolupráci či reagovat na dynamicky se měnící požadavky trhu. COMOS je možné použít jako komplexní řešení nebo použít samostatně jednu z částí COMOS Platform, COMOS Process, COMOS Automation, COMOS Operation nebo COMOS Lifecycle.



Obrázek 2.3: COMOS [4]

moneo (ifm)

Moneo je modulární a flexibilní IIoT platforma, která přináší digitalizaci novým, moderním a jednoduchým způsobem. Kombinuje provozní a informační technologie mimo jiné tím, že data ze senzorů převádí na digitální informace, vyhodnocuje je a přenáší do IT světa. Moneo se skládá kromě základního software i z aplikací např. pro condition monitoring či nastavení parametrů IO-Link senzorů. Do rodiny moneo produktů patří moneo OS, moneo infopoints, moneo edgeConnect, moneo RTM, moneo configure, moneo appliance, moneo starterkit.



Obrázek 2.4: moneo [5]

Jak již bylo řečeno, všechny výše popsané firmy nabízejí komplexní modulární řešení pro plánování a řízení údržby. Kromě toho je možné systém jednoduše rozšiřovat, například ifm nabízí celou škálu unifikovaných chytrých senzorů, které se jednoduše zavádějí. SIEMENS umožňuje propojit systém s rozšířenou realitou, tzv. AR (augmented reality). Vše záleží od konkrétních požadavků zákazníka. Více o ponukách těchto firem je k dispozici na [3, 4, 5].

Jelikož si firmy bedlivě střeží své know-how, je velmi těžké porovnat dané nabídky. Jedním z důvodů je i to, že neexistují žádné trial verze těchto produktů. Dalším důvodem je zmíněná možnost si produkt přizpůsobit. Takže výsledné produkty mohou ve výsledku být ve velmi rozlišné funkční i cenové relaci. Proto se v mnoha případech sahá po vlastních řešeních, které jsou pouze těmito systémy inspirovány.

Kapitola 3

Použité technologie

Tato kapitola popisuje technologie, které budou využity k vývoji systému. Bylo přihlédnuto k současným trendům implementace desktopových aplikací, kde mezi důležité parametry patří snadná rozšiřitelnost, stabilita a nekonfliktnost jednotlivých částí programu.

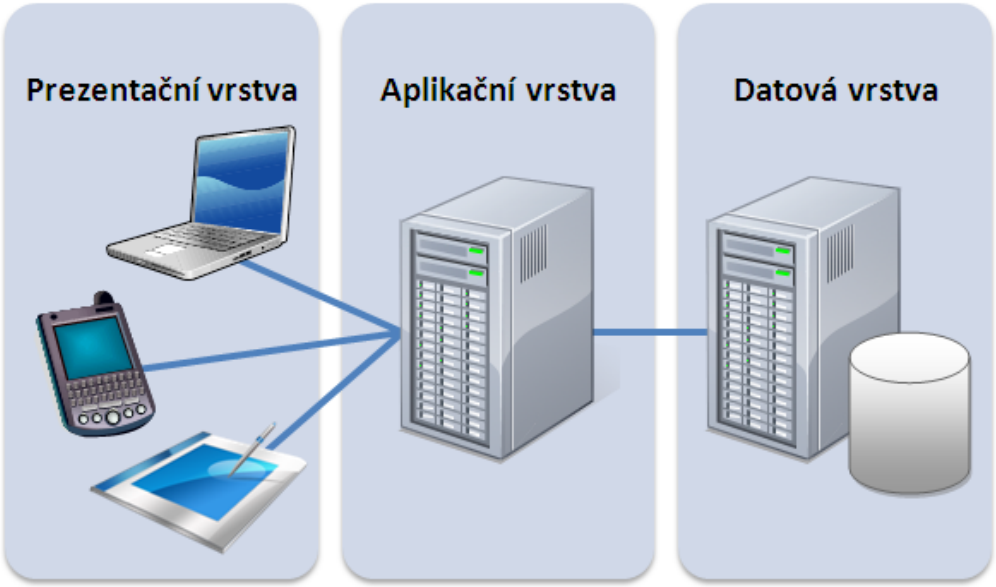
3.1 Klient-server architektura

Architektura klient-server je forma distribuovaného zpracování výpočetního výkonu mezi koncovým zařízením (klientem) a serverem, kteří mezi sebou komunikují a předávají si vzájemně data. Tento model představuje určitý kompromis mezi absolutní centralizací, kdy byly veškeré zdroje sdíleny a absolutní decentralizací, kdy byly veškeré prostředky svěřeny do rukou uživatele. Klient překládá uživatelův požadavek tak, aby byl srozumitelný serveru, čeká od něj odpověď, kterou překládá zpět tak, aby byla srozumitelná klientovi, který ji na obrazovce prezentuje uživateli. Vhodnou volbou komunikace a formulováním dotazů lze výraznou měrou přispět k redukci přenášených dat a tím pádem i k zvýšení rychlosti celého systému. V architektuře klient-server tedy server zpracovává dotazy v databázi a klient je prezentuje, zajišťuje aplikační logiku a zprostředkovává rozhraní pro uživatele.

3.1.1 Třívrstvá architektura

Architektura klient-server se v dnešní době vyskytuje v mnoha formách. Donedávna nejčastěji používaná dvouvrstvá architektura je postupně nahrazována architekturou třívrstvou, u které dochází na straně serveru k oddělení aplikační a datové vrstvy. Uživatel má k dispozici tzv. prezentační vrstvu, která má za účel zobrazovat data uživateli, případně získávat od něj vstupní data. Tato vrstva bývá často realizována jako webová aplikace nebo tenký klient vytvořen v konvenčním jazyce jako Java nebo C#. Aplikační vrstva bývá umístěna na straně serveru a zaštiťuje veškerou logiku aplikace. Tato část systému bývá nejběžněji vytvořena v jazyce JAVA, C#, Python, C++ atd. Poslední vrstvou je datová sloužící k uchovávání informací. Nejčastěji se zde setkáme s DBMS Oracle,

MS SQL, MySql. Postupem času se zvyšuje popularita pronajmutí datového místa v cloudu specializované firmy (např. Microsoft Azure) čímž se mnohonásobně zvyšuje bezpečnost a dostupnost dat.



Obrázek 3.1: Základní schéma 3-vrstvé architektury [6]

Hlavní výhody třívrstvé architektury

- **Rychlejší vývoj** - na jednotlivých komponentách systémů mohou pracovat jednotlivé týmy současně.
- **Škálovatelnost** - tím, že výsledná aplikace je rozdělena do nezávislých celků, lze odhalit a následně posílit nejslabší místo v systému - bottle neck.
- **Snadnější údržba** - jednotlivé funkční celky jsou jednoduše nahraditelné a dá se snadno provést výměna ať už databáze, aplikačního serveru nebo ověřovací služby.
- **Bezpečnost** - data jsou ukládána na specializovaném serveru a přístupná pouze přes definované rozhraní, což je mnohem bezpečnější než na zařízení klienta.

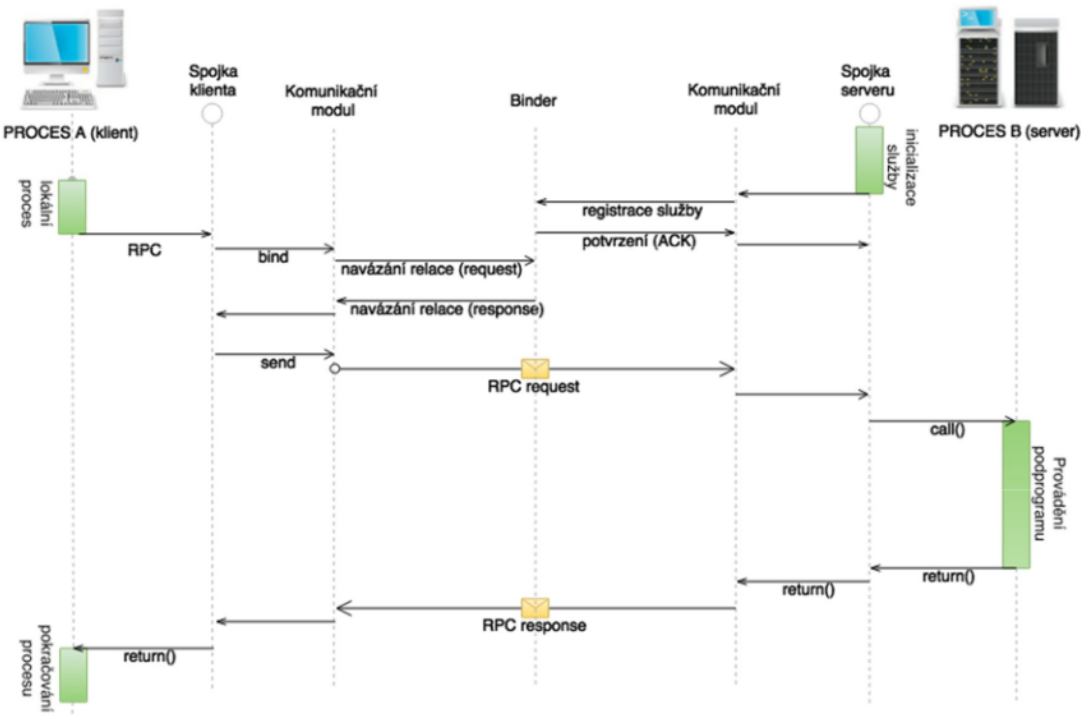
Hlavní nevýhody třívrstvé architektury

- Závislost na kvalitě sítě, tj. pokud není konektivita k síti, nejsou přístupné data a je velmi výrazně omezena možnost používat systém.
- Výrazně dražší zavedení, jelikož je nutné pořídit a udržovat více částí než u architektur jednodušších.

3.1.2 RPC

RPC je jedním ze způsobů komunikace mezi dvěmi procesy v distribuovaném prostředí. Základní myšlenkou RPC je, že program volá funkci, která se nachází na jiném zařízení, stejně jako by volal funkci lokální. Toto volání v ideální formě probíhá tak, že program vytvoří volání funkce, předá parametry a přepne se do stavu čekajícího. Na serverové straně je požadavek příjmu vyřízen a výsledek vrácen volající straně. Čekající program data příjme a dál pokračuje v sekvenčním zpracování.

RPC systémy bývají často navrženy tak, aby byly nezávislé na konkrétním použití programovacího jazyka. Jelikož se mohou tímto lišit jednotlivé lokální datové typy a datové typy ve zprávách, tak se často využívají konvertory dat. Ty bývají nejčastěji součástí takzvaných spojek (stubs) jak na straně klienta, tak na straně serveru. Tudíž program volá klientskou spojku, která připraví zprávu pro volání procedury a připraví parametry do vhodné podoby pro přenos. Tomuto úkonu se říká marshalling nebo též serializing. Tuto zprávu přijme serverová spojka a provede unmarshalling neboli deserializing a zavolá již pro server lokální proceduru, která data zpracuje. Schéma popsané komunikace lze vidět na obrázku 3.2.



Obrázek 3.2: Komunikace dvou procesů pomocí RPC [7]

V rámci RPC vznikl i standard pro popis rozhraní v komunikaci - Interface definition language. Postupem času standardy začaly přibývat, jako příklady uveďme alespoň SOAP, který je založen

na XML nebo AIDL vyskytující se v systémech Android.

3.2 Jazyk C#

C# je moderní objektově orientovaný jazyk, který byl ve verzi 1.0 spolu s .NET Frameworkem v roce 2002 uveden na trh firmou Microsoft. Na svém počátku čerpal z jazyků Java a C++. Jeho podrobnější specifikaci lze nalézt ve specifikaci ECMA [8]. V nynější době je jedním z nejrozšířenějších programovacích jazyků, který lze využít k tvorbě širokého spektra aplikací. Desktopové, serverové, databázové, mobilní ale i webové programy spadají do portfolia tohoto jazyka [9]. V době psaní této bakalářské práce je aktuální verze 9.0.

3.2.1 Technologie .NET Framework

Jedná se o běhové prostředí pro systém Windows, které poskytuje různé služby pro aplikace, které jsou napsané pro .NET Framework. Skládá se z dvou hlavních částí a to modulu CLR (virtuálni stroj, který mimo jiné řeší správu paměti) a rozsáhlých třídních knihoven, které obsahují otestovaný kód připravený k znovupoužití. Co se týče současného stavu, je poslední verzí .NET Framework 4.8, která je již poslední, jelikož Microsoft se dále rozhodl vyvíjet pouze .NET Core a .NET Standard, které jsou nezávislé na platformě.

3.2.2 WPF

Hojně využívaný grafický framework na tvorbu okenních aplikací. V minulosti nahradil starší WinForms a přinesl zcela nové možnosti jako nezávislá grafika na DPI, propracovaný data binding (provázání dat mezi prezentační a view model částí), možnosti animací nebo také podporu 3D grafiky prostřednictvím DirectX. Vlastní okna jsou psána pomocí jazyka XAML, který je založen na XML. Tvorba uživatelského prostředí pomocí tohoto jazyka je velmi intuitivní a rychlá [10].

Byl představen v rámci .NET Frameworku v roce 2006. I když v dnešní době pomalu nastupuje UWP (Universal Windows Platform), který má podporu pro mnoho různých platforem (Xbox, mobilní zařízení atd.), je stále pro vývoj desktopových aplikací doporučeno využívat WPF.

3.2.3 MVVM

Představuje velmi populární návrhový vzor - Model-View-ViewModel. Tento vzor dovoluje oddělit logiku aplikace od uživatelského rozhraní. Tomu je uzpůsobeno i WPF, které nabízí bindings a commands jako náhradu uživatelského rozhraní, které je řízené událostmi. Hlavní myšlenka je mít třídu, která si drží stav aplikace - ViewModel. Z této třídy čerpá informace prezentační vrstva View. Poslední vrstvou v této architektuře je Model, který reprezentuje třídy umožňující přístup k datům v persistentním uložišti, nejčastěji v databázi. Změny ve ViewModelu jsou poté po změně dat propagovány vyvoláním události implementující rozhraní *INotifyPropertyChanged*.

3.2.4 WCF

Jedná se o komunikační framework umožňující vytvářet servisně orientované distribuované aplikace. Je přímým nástupcem DCOM a .NET Remoting a byl představen spolu s .NET Frameworkem 3.0. Mezi hlavní rysy WCF patří sjednocení předchozích technologií, interoperabilita napříč platformami a orientace na služby [11]. Základním stavebním kamenem jsou služby, které se skládají z třídy služby, hostovacího prostředí a jednoho či více endpointů. Endpoint je jedna z nejdůležitějších částí samotné služby, jelikož přes něj probíhá veškerá komunikace s klientem. Často bývá popisován jako tzv. ABC (Address, Binding, Contract).

- **Address** - určuje, kde se endpoint nachází (kam na síti mají přicházet zprávy, které endpoint přijímá).
- **Binding** - určuje způsob komunikace, tj. komunikační, transportní a kódovací protokol.
- **Contract** - definuje operace poskytované endpointem.

3.3 MS SQL Server

Při výběru konkrétního databázového systému přicházely k úvahu DBMS Oracle anebo MS SQL Server od firmy Microsoft. Jelikož byl jako programovací jazyk zvolen C#, přednost dostal produkt od stejné firmy. MS SQL Server je relační DBMS podporující objektové relační datové modely, který je vyvíjen společností Microsoft a je postavený na architektuře klient-server. Systém je založený na jazyce SQL a podporuje procedurální jazyk T-SQL. Mezi další výhody patří škálovatelnost, bezpečnost, robustnost, podpora transakcí, XML a JSON. Microsoft nabízí integrované prostředí SQL Server Management Studio, které disponuje řadou funkcí a nástrojů, které lze použít při vývoji a správě databází. Umožňuje například monitorování a diagnostikování pomalu běžících dotazů a jejich krokování k nalezení příčiny problému. Současné verze jsou dostupné i pro platformu Linux. Poslední vydanou verzí je Microsoft SQL Server 2019. Co se týče verzí MS SQL Serveru jsou k dispozici na trhu 4 varianty: Enterprise, Standard, Express a Web [12]. Jednotlivé verze se kromě ceny liší hlavně v klíčových faktorech jako cena, maximální počet využitých jader procesoru, paměti a také velikosti databáze (viz tabulka 3.1).

Tabulka 3.1: MS SQL Server

Vlastnost \Edice MS SQL Serveru 2019	Enterprise	Standard	Express	Web
Maximální počet jader CPU	OS Max	24	4	16
Maximální využitelná paměť RAM	OS Max	128 GB	1 GB	64 GB
Maximalní velikost databáze	524 PB	524 PB	10 GB	524 PB

Dle odhadované velikosti databáze, požadavků na rychlost a výpočetní výkon byla finálně zvolena varianta MS SQL Express 2019, která je distribuována zdarma a plně vyhovuje všem požadavkům.

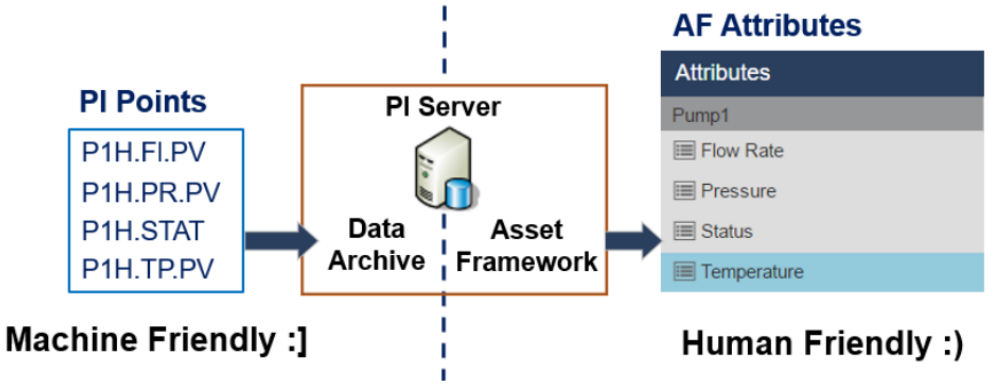
3.4 PI Systém

PI Systém je program, který v reálném čase zpracovává produkční data. Vyvinula jej firma OSIsoft v roce 1980 a nabízí jej doposud jako svůj jediný produkt. Tudíž se jedná o 40 let vylepšovaný produkt, který je špičkou ve své oblasti a patří k těm nejrozšířenějším. Nejčastější odvětví, kde PI Systém nalezneme jsou ty, které zpracovávají obrovské množství informací jako energetika, chemický či těžařský průmysl. V dnešní moderní době Průmyslu 4.0, kdy je hlad po zpracování informací a všeobecné digitalizaci se PI Systém dostává i do výrobních podniků jako ocelářství a farmacie, či do různých datových center. PI Systém je real-time data historián, tj. software, který je schopen zpracovávat a hlavně ukládat velké množství dat. Pro tento účel má vyvinuté své vlastní komprimační a aproximační metody nad NoSql databází a dokáže tak poskytovat data k analýzám nejen z dat aktuálních, ale i z hluboké historie. Tento produkt je plně placený (OSIsoft nenabízí žádnou verzi zdarma ani trial verzi), kde jedním z hlavních kritérií je počet tagů, ve kterých se budou informace uchovávat. Také se zvlášť dokupují moduly, případně místo v cloudovém uložišti. Nedílnou součástí je také roční podpora, v rámci které má uživatel nárok na rady s nastavením systému i na jeho aktualizace.

3.4.1 Jak PI Systém pracuje

V prvé řadě jde o sběr dat z rozličných vstupů. Může se jednat o jednotlivé senzory, IoT zařízení, ale také zdrojem mohou být různé již fungující systémy. Je nabídnuta celá řada rozhraní (v kolekci je přes 400 různých) a konektorů, takže se dá říct, že cokoliv může sloužit jako vstup. Mezi základní konektory patří PItoPI, OPC, Modbus, RDBMS nebo UFL (univerzální nahrávač souborů). Jednotlivá data jsou uložena v tzv. PI Data Archívech. Ty bývají sdružené do tzv. Collective memberů (lze pochopit jako zdvojené uložení) a tím pádem PI Systém nabízí velkou míru ochrany dat proti výpadku a tím i vysokou dostupnost. Další službou je postupná komprimace dat, kde se historické data nahrazují pouze klíčovými hodnotami. Díky této komprimaci je možné uchovat delší časový úsek dat ve víceméně nezměněné podobě. Dále jsou data zpracovávána Asset Frameworkem, kde se k surovým datům přidávají různá meta data (informace o kvalitě dat nebo jednotky). Framework umožní data grupovat, spouštět se různé analýzy nebo vyvolávat Event Framy (tj. události spuštěné při splnění podmínek).

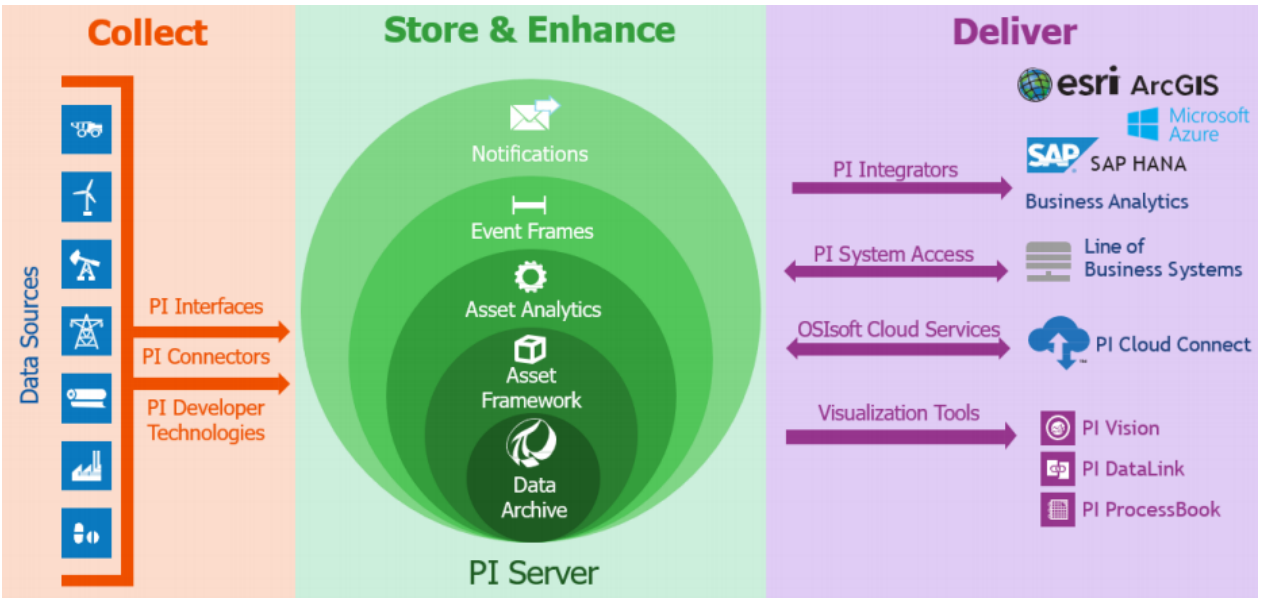
V neposlední řadě přichází vlastní prezentace a vizualizace dat. Lze využít různé interface do systémů vyšší úrovně jako SAP nebo pořád oblíbené exporty do Excelu pomocí PI Data Linku. PI Process Book je aplikace, která dává uživatelům možnost analyzovat nebo si lokálně vizualizovat data. Nástroj PI Vision (dříve PI Coresight) je další možnost jak vytvořit náhled na data, výsledky analýz, různých notifikací i člověku, který nemá s programováním zkušenosti. Tento náhled poté umístit na webu a tím pádem vizualizace sdílet s ostatními uživateli. Pro programátory je připraveno



Obrázek 3.3: Převod dat z OPC do Asset Frameworku [13]

několik nástrojů, pomocí kterých lze importovat data z PI Systému do vlastních aplikací. K dispozici je :

- **PI Web API** - nabízí RESTful interface do PI Systému, tj. dává možnost klientským aplikacím zapsat nebo číst informace prostřednictvím HTTPS.
- **PI SDK** - jedná se o knihovnu, která podporuje objektově orientovaný přístup ke čtení a zápisu dat. Celý balíček obsahuje dokumentaci, podpůrné soubory nebo ukázkové kódy.
- **PI AF SDK** - jde o novější zpracování vývojářského balíčku (knihovny), která kromě přístupu k datům PI Data Archívu umožňuje pracovat i s AF serverem.



Obrázek 3.4: Základní schéma fungování PI Systému [13]

Kapitola 4

Analýza a sběr požadavků

V dnešní době se v nejednom podniku razí heslo Průmysl 4.0, který přináší větší digitalizaci procesů s důrazem na maximalizaci úspor a minimalizaci jakéhokoliv plýtvání (jak lidských, tak hmotných zdrojů). Je důležitá snaha minimalizovat prostoje z důvodů kritických výpadků provozních zařízení, kde takovýto prostoj lze vyčíslit i na miliony korun. Také je důležité mít přehled o veškeré plánované údržbě z důvodu efektivního využití všech pracovníků.

Proto padlo rozhodnutí vytvořit aplikaci, která by umožňovala plánování údržby na základech CbM. Bylo využito spolupráce s výrobní firmou, která byla ochotna poskytnout provozní data a konzultaci s pracovníky údržby. To umožnilo zanalyzovat stávající stav a také inspirovat se reálnými požadavky, aby byla výsledná aplikace jednoduše použitelná v praxi.

4.1 Stávající stav

V současnosti je údržba plánována pouze ručně na papíře nebo v programu MS Excel mistry technických úseků. Je zde sice opora v PI Systému, který obsahuje data o místech, pro které má být údržba vykonávána, včetně jejich intervalů údržby, avšak tento způsob se jeví být nevhodný. PI Systém by se měl používat primárně pro uchovávání a analýzu dat a ne pro řízení činností. Velká nevýhoda současného stavu je v tom, že plánování veškerých údržbářských prací je plně v kompetenci lidí, bez opory sofistikované výpočetní techniky a pouze na úrovni lokálního pracoviště. Dále se také při údržbě nezohledňují faktory jako skutečná doba provozu zařízení. Tudíž se stávalo, že některá zařízení se dostala do údržbového plánu i když byla dlouhodobě vyřazena z provozu a tím vznikala ztráta jak času tak financí. Je zde i nevyhovující evidence, kdo a kdy provedl údržbu na daném zařízení. V případě jeho výpadku je složité a časově náročné dohledat záznamy z různých papírů a nejednotných listů MS Excelu.

V minulosti proběhly pokusy z řad zaměstnanců vytvořit jednoduchý systém pro řízení údržby v rámci existujícího PI Systému. A přes to, že tento způsob již využívá skutečné parametry a chod zařízení, potýkal se s nedostatky jako :

- Nutný přímý zápis do PI Systému (ručně pomocí PI Exploreru či pomocí MS Excelu a PI DataLinku).
- Zápis mohl být prováděn pouze lidmi, kteří měli povolení zapisovat do PI Systému a těch nebylo mnoho. Samozřejmě nebylo řešení dát přístup na zápis všem.
- Evidence založená čistě na PI Systému byla poměrně nepružná, pomalá (hlavně vinou mnoha analýz a Event Framu) a směřovala také k vyčerpání volného počtu tagů (PI Systém se licencuje na množství tagů - koncových míst).

Pokud tedy shrneme výchozí stav, tak máme v současné době k dispozici:

- Seznam jednotlivých údržbových míst, jejichž struktura již navedena v systému PI.
- Informaci o skutečné době provozu zařízení - motohodiny, které jsou v PI Systému pomocí analýz vypočítány na základě signálu z OPC (např. otáčky nebo proudové odběry motoru).
- Informace o intervalech údržby, které jsou získány z oficiálních dokumentů výrobce nebo na základě zkušeností techniků.

4.2 Výsledné očekávání

Na základě schůzek s jednotlivými pracovníky údržbářských týmů a jejich vedení, dále také dle dostupné teorie CbM byly vybrány tyto následující klíčové body budoucí aplikace. Předpokládá se, že pokud se podaří splnit tyto body, bude výsledná aplikace zjednodušením a zpřehledněním organizační práce při plánování údržby.

- Vytvořit jednotnou aplikaci pro použití na kterémkoliv oddělení a tím ulehčit zaškolování pracovníků, případně umožnit jejich zastupitelnost.
- Zrychlit a zjednodušit zadávání záznamů o údržbě do systému.
- Zredukovat v PI Systému statické data o údržbách a přenést je do relační databáze, která je na tyto účely vhodnější a levnější. Tímto uvolnit místo v PI Systému provozním informacím, pro které je určen.
- Automaticky plánovat údržbu na základě skutečné provozní doby zařízení. Umožnit vedoucímu pracovníkovi daného oddělení ruční zásah do tohoto plánu.
- Zpřístupnit jednotlivým pracovníkům údržby přehledný plán údržby, včetně informací, které jsou pro tento úkon nezbytné.
- Odstranit nutnost instalace systému PI na počítač pracovníků údržby a také eliminovat nutnost zavádění přístupových práv na zápis.
- Pro management mít možnost získat informace a reporty o provedených údržbách.

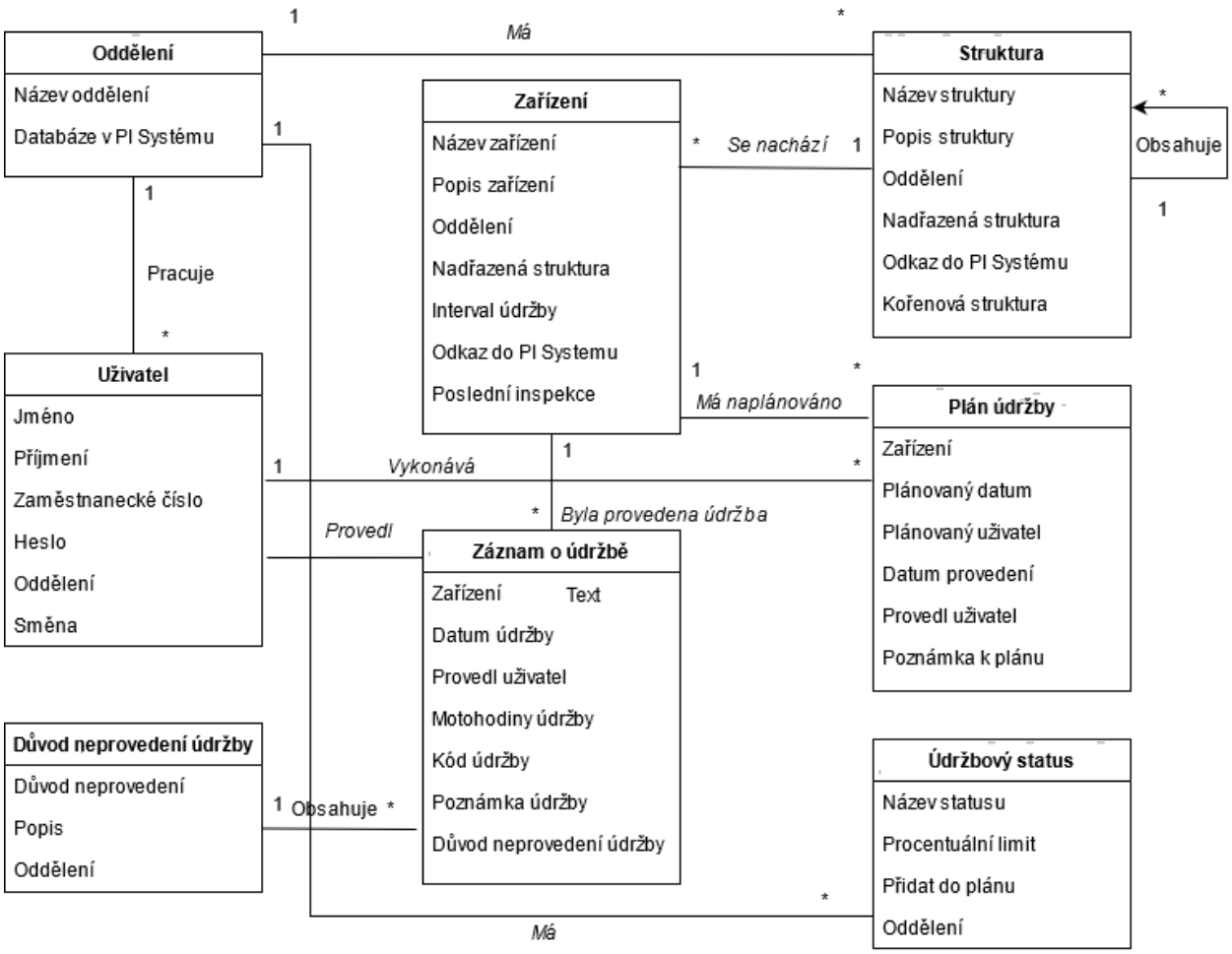
4.3 Entity systému

Hlavním úkolem aplikace bude sledování provozu jednotlivých **zařízení**, které na jednotlivých **odděleních** jsou součástí větších celků, tj. **struktur**. Každé zařízení má svůj interval údržby, tj. čas po jehož uplynutí by měla být vykonána údržba. Na základě provozní doby bude automaticky nebo ručně možno vytvářet **plány** údržby, které budou přidělovány konkrétním **pracovníkům**. Pokud pracovník údržbu vykoná, provede zápis formou **provedené údržby**. Interval údržby tím pádem začne běžet znova až do doby, kdy bude nutné znovu zařadit do plánu. To zda je zařízení zařazeno do plánu se určuje na základě **údržbového statusu**. Ten se počítá jako procento již uběhnutého intervalu údržby. Někdy se může stát, že i když je údržba naplánována a pracovník jí chce provést, není mu to z různých důvodů umožněno. Tyto důvody neprovedení inspekce jsou předem definovány, pokud jsou vyplněny, není resetován interval údržby. Pracovník tím ale dává vědět, že se o údržbu pokusil.

Z tohoto krátkého popisu vyplývá, že bude nutné evidovat tyto základní entity:

- **Oddělení** - entita, která udržuje základní informace o oddělení, odkud čerpá motohodiny a další důležité nastavení hlavně pro komunikaci s PI Systémem.
- **Struktura** - entita, která bude obsahovat informace o struktuře daných zařízení, bude uložena hierarchicky a bude z ní možno vytvořit strom jednotlivých zařízení.
- **Zařízení** - entita, sloužící k uchování informací o jednotlivých zařízeních, především statická data.
- **Záznam o údržbě** - dynamická data o jednotlivých provedených údržbách. Zde je důležitý kód údržby, který říká kolikátá údržba byla provedena, tj. každou provedenou údržbou se navyšuje o 1. Dále je nutné uchovat při jakých motohodinách byla údržba vykonána (tento údaj, celkové motohodiny a interval údržby poté slouží k výpočtu stavu zařízení).
- **Plán údržby** - entita, která slouží k reprezentaci jednotlivých plánů a také k zaznamenání jejich provedení.
- **Údržbový status** - entita, která popisuje stav zařízení. Má procentuální limit a příznak, zda zařízení s daným statusem zařadit do plánování údržby.
- **Důvod neprovedení údržby** - vyjadřuje stavy provozu, kvůli kterým nemohla být údržba provedena.
- **Uživatel** - entita, která bude udržovat základní informace o pracovníkovi.

Důkladnější pohled na základní entity budoucího systému nám poskytne diagram 4.1.
Se systémem budou pracovat následující typu uživatelů :



Obrázek 4.1: Základní entitní diagram

- **Administrátor** - role, která může nastavit údaje pro všechny oddělení - uživatele, zařízení i plány.
- **Vedoucí** - vedoucí pracovník. Ten bude mít přiděleno své oddělení (jedno oddělení může mít více vedoucích pracovníků), kde v rámci svého oddělení může nastavit libovolné data uživatelů a může měnit automatické plány. Má přístup k jednotlivým reportům svého úseku.
- **Operátor** - pracovník, který provádí jednotlivé údržby. Může vidět celkový stav zařízení svého oddělení, může vyplnit záznam o údržbě, případně si bude moct nastavit heslo.

4.4 Analýza procesů

Základní procesy a operace jednotlivých rolí zobrazí následující kompletní Use Case diagram 4.2, kde uživatelé Administrátor, Vedoucí a Operátor budou používat aplikaci desktopovou. Budou se muset přihlásit, aby se dalo zaznamenávat kdo provedl dané úkony. Operátor bude mít v aplikaci nejnižší práva a kromě změny svého hesla bude moci zadat pouze záznam o údržbě, případně si zkontrolovat svůj plán práce nebo svoje výsledky. Vedoucí daného oddělení bude moci navíc oproti operátorovi i upravovat plán práce, měnit data o uživatelích včetně resetování hesla, nastavit základní číselníkové hodnoty (statusy a důvody neprovedení údržby). V neposlední řadě bude mít přístup k výstupním sestavám a grafům. Administrátor bude mít kontrolu nad všemi údaji v aplikaci, včetně možnosti ovlivňovat data jakéhokoliv oddělení.

Systémem se rozumí aplikační server, který bude zastávat dvě hlavní funkce. První bude poskytování dat všem klientským aplikacím. Druhou funkci bude mít v roli aplikačního serveru, kde bude v periodických intervalech aktualizovat motohodiny jednotlivých zařízení, vyhodnocovat stavy a případně plánovat údržbu pro zařízení.

Procesy jako přihlášení uživatele, změna hesla, vypsání reportu atd. považuji za běžně používané, proto je zde není nutné více rozepisovat. Důležité je však pozastavit se u tří procesů, které již konkrétní popis vyžadují a to zadání údržby, zadání neprovedené údržby a automatický plán údržby.

4.4.1 Zadání údržby

Jedná se o proces, kdy je vložen záznam o údržbě do systému. Základními vstupními informacemi jsou dané zařízení, datum provedení údržby, případně poznámka. Hlavní podmínku, kterou musí systém ověřit je, že se záznam nevkládá již před provedenou údržbu. Vkládání záznamu do budoucnosti ověří klientská aplikace.

1. Uživatel vybere zařízení, na kterém se má provést údržba.
2. Uživatel vyplní datum provedení údržby, poznámku údržby (pokud má nějakou).
3. Uživatel potvrdí vložení údržby.

Obrázek 4.2: Use case diagram

4. Systém načte poslední provedenou údržbu.
5. Systém zkontroluje, zda se nevkládá údržba před datum provedení poslední údržby. V případě, že datum vkládané údržby je starší než poslední provedená, skončí se chybovou hláškou.
6. Systém zjistí kolik mělo zařízení motohodin. Provede dotaz do systému PI. Pokud nenajde žádné data, údržbu neuloží, jelikož zařízení nebylo daný datum ještě uvedeno do provozu.
7. Systém navýší kód údržby o 1.
8. Systém uloží záznam o údržbě. V případě úspěchu také uloží časové razítko provedené údržby do PI Systému.
   (a) Pokud na dané zařízení byl vytvořen plán, systém jej označí za splněný a poznamená se datum provedení a osoba, která údržbu provedla.
   (b) Pokud na daném zařízení nebyl vytvořen plán, systém vytvoří v plánu záznam o údržbě mimo plán. U tohoto záznamu systém vyplní datum provedení údržby a osobu, která údržbu provedla.
9. Systém provede znovu načtení provozních dat a vypočítání aktuálního stavu zařízení.

4.4.2 Zadání neprovedení údržby

Jedná se o proces, kterým pracovník zaznamená neúspěšný pokus o provedení údržby. Na rozdíl od běžného záznamu o údržbě se musí zadat i důvod, kvůli kterému nebylo možné údržbu provést. Opět proběhne kontrola na datum, který nesmí být z budoucnosti a také nesmí byt starší než poslední údržba. V tomto případě se nenavýší kód údržby, není také splněn plán (pokud byl naplánován) a tento typ záznamu se také nezapisuje do PI Systému.

1. Uživatel vybere zařízení, na kterém se měla provést údržba.
2. Uživatel vyplní datum pokusu o provedení údržby, poznámku údržby (pokud má nějakou).
3. Uživatel musí vybrat důvod neprovedení údržby.
4. Uživatel potvrdí vložení záznamu.
5. Systém načte poslední provedenou údržbu.
6. Systém zkontroluje, zda se nevkládá údržba před datum provedení poslední údržby. V případě, že datum vkládané údržby je starší než poslední provedena, skončí se neuložením záznamu.
7. Systém zkopíruje motohodiny a kód údržby z poslední provedené.
8. Systém uloží záznam (do PI Systému se neukládá nic).
9. Systém provede znovu načtení provozních dat a vypočítání aktuálního stavu zařízení.

4.4.3 Automatický plán údržby

Tento proces bude zajišťovat aplikační server. Pro všechny zařízení se v pravidelných intervalech načte ze systému PI aktuální stav motohodin, pomocí dat z poslední údržby (při kolika motohodinách se údržba prováděla) se vypočte, kolik času uběhlo od poslední údržby. Spočte se z kolika procent je naplněn interval údržby. Poté se vyhodnotí jaký stav se zařízení přiřadí. Pokud daný stav má nastaven atribut *Přidej do plánu* nebo do vypršení intervalu zbývá méně než 24 hodin, zařízení se zařadí do plánu na údržbu. Tam je samozřejmě zařazeno jen tehdy, pokud neexistuje nesplněný plán pro dané zařízení.

1. Systém zakáže provádět ostatním uživatelům změny během plánování.
2. Systém provede pro všechna zařízeni daného oddělení.
   (a) Systém provede aktualizaci motohodin z PI Systému.
   (b) Systém vyhodnotí nový stav zařízení.
   (c) Pokud stav odpovídá stavu, který se má zařadit do plánování nebo zbývá do konce údržbového intervalu méně než 24 hodin, tak se provede následující.
      i. Systém zkontroluje, zda zařízení nemá aktivní plán, pokud ano ukončí se zpracování tohoto zařízení.
      ii. Pokud není aktivní plán určí se datum, na kterou je údržba plánována, tj. 24 hodin před vypršením intervalu.
      iii. Systém určí směnu, která vychází na daný datum a dle toho vybere pracovníka z řad operátorů, který má na daný den nejméně naplánovaných akcí.
      iv. Pokud nenajde pracovníka, vybere Vedoucího pro dané oddělení.
      v. Systém uloží záznam o naplánované údržbě.
   (d) Systém vybere další zařízeni a vrací se k bodu a).
3. Systém povolí provádět změny na aktualizovaných zařízeních.

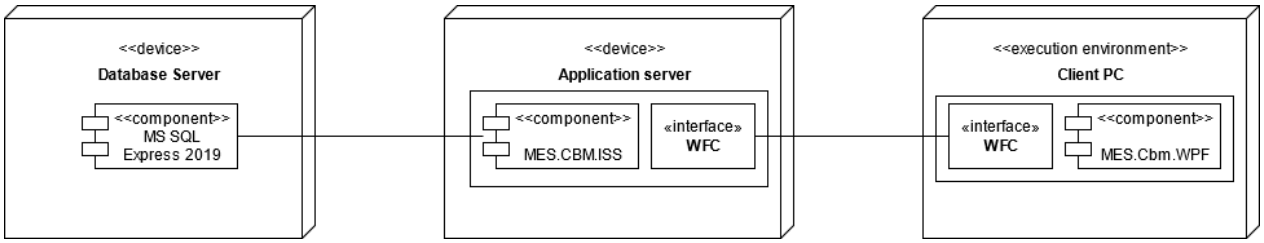
Kapitola 5

Implementace a nasazení

Tato kapitola bude obsahovat konkrétní návrh a implementaci aplikace. Nejprve popis výběru architektury, pak detailněji její jednotlivé části. Do přílohy A byl umístěn detailní postup k uvedení aplikace do provozu. Na vývoj celé aplikace budou použity nástroje firmy Microsoft, konkrétně tedy Microsoft Visual Studio 2019 a pro uložení dat MS SQL Server 2019 Express.

5.1 Implementace systému

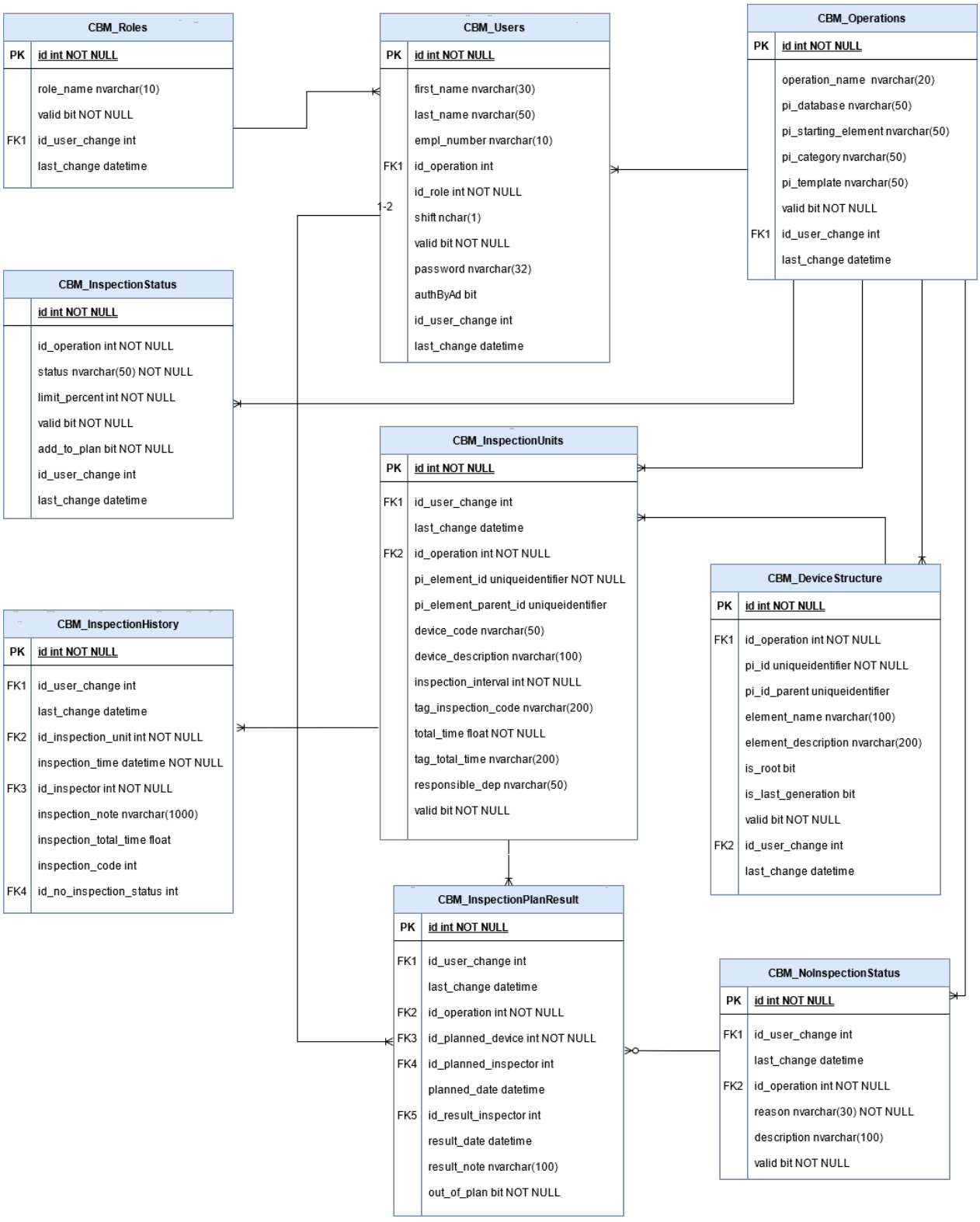
Pro implementaci veškeré potřebné funkcionality byla zvolena architektura klient-server ve tří-vrstvé variantě. Bylo tedy potřeba připravit vrstvu datovou, aplikační server, který posloužil jako business vrstva, a také vrstvu klientskou, která bude reprezentována desktopovou aplikací. Pro přehled je schéma nasazení aplikace zobrazeno na obrázku 5.1.



Obrázek 5.1: Diagram nasazení

5.1.1 Datová vrstva

Vlastní databáze byla vytvořena pomocí nástroje MS SQL Server Management Studio. Byly vytvořeny základní tabulky, které odpovídají entitám ze systému. Všem tabulkám byl zvolen prefix CBM. Každá tabulka je opatřena automaticky generovaným primárním klíčem Id, dále je v tabulce vždy použit sloupec id_user_change, který určuje uživatele měnícího záznam naposledy. Poslední



Obrázek 5.2: ER diagram

společný sloupec všech tabulek je last_change uchovávající datum a čas poslední změny. Většina tabulek také obsahuje sloupec valid, který signalizuje zda je záznam platný. Je zde z důvodu realizace mazání záznamů, kdy záznam fyzicky v databázi zůstane, ale program jej ignoruje (nepoužívá), leda uživatel o to explicitně požádá. Kompletní ER diagram je zobrazen na obrázku 5.2. Po vložení základních údajů do tabulek byly pomocí Management Studia vytvořeny skripty, které umožní databázi exportovat (tabulky a data).

Dále vznikly tabulkové entity, které slouží jako základ k business entitám a jsou také využívány pro komunikaci mezi aplikačním serverem a klientem (pro serializaci a deserializaci). Mají prefix T, např. TCbmUser a obsahují pouze datové vlastnosti (property). Aby šlo tyto entity používat v rámci celé aplikace, byly vyčleněny do samostatného projektu. Co se týče business entit, ty dědí datové položky z tabulkových entit, dále přidávají vlastní funkcionalitu jako konverzi na tabulkovou entitu, načtení entity z databáze nebo metodu AddOrUpdate, která na základě stavu entity provede uložení či změnu dat. To v jakém stavu se entita nachází se jednoduše pozná podle Id. Pokud není vyplněno, jedná se o entitu neuloženou, tedy novou.

Pro propojení tabulek a business vrstvy byl použit návrhový vzor *Repository* [14], díky kterému je datová vrstva plně odstíněna od business vrstvy a dává tak volnější ruce použít databázi jiného výrobce s minimálním zásahem do celkové aplikace. Pro každou entitu byla vytvořena oddělená třída, která dědí z třídy BaseRepository. Ta implementuje základní CRUD operace a velmi tak zjednodušuje finální návrh. Kromě dědičnosti je zde použito implementování rozhraní každé repository a tím se více odstíní vlastní implementace. Jako ukázka je uvedena metoda pro vytvoření záznamu z bázové repository.

```
public long Create(T2 item)
{
   using (SqlConnection connection = new SqlConnection(ConnectionString))
    {
      try
        {
        connection.Open();
        SqlCommand cmd = new SqlCommand(CmdCreate, connection);
        PrepareCommand(cmd, item, false);
        var result = cmd.ExecuteScalar();
        return long.Parse(result.ToString());
        }
        catch(Exception ex)
        {
          logger.ErrorFormat($"Create ({nameof(T2)}) : Record was not created.
              Reason {ex.Message}.");
            return 0;
```

```
        }
    }
}
```

Listing 5.1: Metoda base repository pro vytvoření záznamu.

5.1.2 Business vrstva

Tato vrstva byla implementována jako Windows služba s názvem **ISS** - *Infra Structure Server*. Jako kostra této části byla využita implementace WCF služby ServiceGatewayServer, jejíž popis ale není předmětem této práce. Hlavní úlohou služby je vytvořit kontejner instancí třídy *CbmOperationServer*, které mají za úkol inicializovat, zpracovávat a poskytovat živá data daného oddělení klientům. Pro každé oddělení tedy vznikne jedna instance. Dále je inicializován *ServiceGatewayServiceHost*, což je implementace serverové části WCF komunikace, která má za úkol poskytovat klientům funkce pro načítání a ukládání dat do databáze, případně dále pracovat s business objekty. Co se týká konfigurace služby, ta je nastavitelná pomocí souboru **confing.ini**. Mezi nejdůležitější nastavení patří připojovací řetězec k databázi, časový interval, ve kterém probíhá vykonávání periodických akcí, dále nastavení základních parametrů WCF komunikace - port, cestu, název assembly s interfacem a vlastními poskytovanými službami. Posledním zajímavým nastavením je typ prostředí, kde si lze vybrat volbu **prod** a **test**. Ve volbě test nebude služba provádět veškeré operace s PI Systémem, takže ji lze pustit i bez konektivity.

Služba ISS je navržena tak, aby byla spustitelná jako konzolová aplikace, což se využije v případě ladění programu, tak i jako windows služba, samozřejmě po instalaci. V obou případech je zde implementováno logování provozu pomocí knihovny **Log4Net** [15], která má nastavený výstup do souboru **log.txt**. V případě, kdy je spuštěna v konzolovém módu, je logování v barevném provedení vyvedeno i do vlastního okna příkazové řádky. Službu je nutné spouštět s administrátorskými právy.

Vlastní inicializace serveru pro oddělení pak probíhá v následujících krocích:

1. Server si načte připojovací údaje pro dané oddělení.
2. Server se připojí k PI Systému, načte strukturu z databáze a ověří proti aktuálním datům z PI Systému, zda nedošlo ke změnám. Případné změny se promítnou uložením do databáze.
3. Stejný proces, tj. synchronizace dat, proběhne i pro jednotlivá zařízení daného oddělení, každé zařízení se doplní o cestu v rámci této struktury.
4. Jakmile skončí synchronizace, provede se vytvoření stromové struktury a ze systému PI se aktualizují motohodiny.
5. Pokud veškerá inicializace serveru proběhla v pořádku, je server označen za inicializovaný a začíná jednak poskytovat data a také se v pravidelném intervalu stará o aktualizaci motohodin.

Synchronizaci jednotlivých struktur a statických dat zařízení provede poté po půlnoci, kdy se očekává menší provoz.

Jednotlivé servery jsou udržovány ve třídě *CbmServerContainer.cs*, která je implementována pomocí návrhového vzoru Singleton, přesněji jeho thread safe variantou [14]. To je využito proto, aby data byla přístupná z jakékoliv části business vrstvy.

Aby mohla služba komunikovat s PI Systémem, musí být na daném počítači nainstalováno základní prostředí pro PI. Lze si představit jako nutné knihovny, seznam PI Serverů - Datové a AF Servery. Další nutnou podmínkou je, aby služba byla spuštěna pod účtem, který má právo se systémem PI pracovat. Pro konkrétní načítání a zápis dat do systému PI byla v business vrstvě napsána třída **CmbPiServices.cs**. Následující ukázka kódu naznačí, jakým způsobem se připojujeme k datům v PI.

```
public PIServer ConnectToDataArchive()
{
    try
    {
        PIServers servers = new PIServers();
        PIServer dataArchive = servers.DefaultPIServer;

        var credential = new NetworkCredential("UserName", "UserPass");

        dataArchive.Connect(credential, PIAuthenticationMode.
            WindowsAuthentication);

        return dataArchive;
    }
    catch (Exception e)
    {
        logger.ErrorFormat($"ConnectToDataArchive: Error during connecting
            to PI Archive. Reason {e}.");
        return null;
    }
}
```

Listing 5.2: Metoda pro připojení k PI Data Archívu

Nejdůležitější funkce serveru pro každé oddělení je udržovat reálná, aktuální data. Nad těmito daty dále provádět výpočty, které zjistí v jakém stavu jsou daná zařízení, a v případě potřeby zařadit toto zařízení do plánu údržby. Stručně lze proces popsat tak, že pro všechny zařízení se postupně

provede aktualizace motohodin ze systému PI. Poté se provede výpočet, kolik hodin je zařízení v provozu od poslední údržby. Výsledkem je procentuální hodnota naplnění intervalu údržby, který se porovná s údržbovými statusy pro dané oddělení. Výsledkem je aktuální status pro zařízení, u kterého se zkontroluje, zda toto zařízení zařadit do plánu. Do plánu je zařazeno také zařízení, kterému zbývá do údržby posledních 24 hodin. Údržba se plánuje vždy na začátek směny nejpozději 24 před vypršením intervalu údržby. Poté se vybere seznam pracovníků, kteří na dané směně pracují, porovná se, který má nejméně úkonů a tomu se daný plán údržby přiřadí. Pokud není žádný pracovník na danou směnu definován, je plán přiřazen Vedoucímu daného oddělení. Ten poté může přeplánovat tento úkon někomu ze svých podřízených pomocí klientské aplikace. Jednotlivé plány se tedy zadávají vždy k začátku směny, pořadí jednotlivých úkonů si již určí pracovník. Zde je uveden algoritmus na výpočet směny pro plánované datum. Tato metoda slouží k určení směny ve 4 směnném 8 hodinovém provozu. Jednotlivé směny se střídají na základě klíče 2x ranní, 2x odpolední, 2x noční a dva dny volno.

```
public static string GetShiftFromDate(DateTime dtDate)
{
    string szShifts = "ACBDBADBACADCADBDCBDCACB";
    return szShifts[((int)((dtDate - new DateTime(1970, 1, 6, 6, 0, 0)).
        TotalDays * 3.0)) % 24].ToString();
}
```

Listing 5.3: Metoda výpočtu směny v daný datum.

5.1.3 Klientská vrstva

Klientská aplikace byla připravena pomocí jazyka C# a technologie WPF. Jako návrhový vzor byl zvolen MVVM, který oddělil modelovou část od části prezentační. Díky této zvolené strategii bude do budoucna poměrně jednoduché prezentační vrstvu změnit například na webovou aplikaci reprezentovanou ASP.NET stránkami. Celá aplikace byla nastylována dle moderních zvyklostí a byla zde přidána možnost zapnout si **tmavý režim**, který se dnes houfně zavádí ve všech aplikacích napříč všemi platformami (ten si uživatel zapíná v nastavení systému).

Struktura projektu klientské aplikace je rozdělena do jednotlivých složek:

- **Views** - obsahuje veškeré XAML soubory pro jednotlivé uživatelské komponenty a dialogové okna. Pro rozlišení byla použita předpona Uc pro uživatelské komponenty a Dw pro dialogová okna, která slouží především k zadávání dat.
- **ViewModels** - obsahuje třídy reprezentující data kontexty pro veškeré komponenty a okna. Veškeré třídy pro view modely oken dědí ze třídy *VmBase.cs*, která implementuje důležitá rozhraní jako *INotifyPropertyChanged* nebo *IDataErrorInfo*. Dále jsou zde implementovány

funkce pro validaci vstupních dat, metoda pro zavření okna a základní *Command* příkazy pro ovládání oken.

- **Services** - zde jsou umístěny třídy reprezentující spojky (konektory) na aplikační server.
- **Styles** - zde jsou umístěny resource dictionary veškerých stylů použitých v aplikaci. Tyto jednotlivé styly jsou poté spojeny v souboru *app.xml* do jediného stylu.
- **Helpers** - složka obsahuje pomocné třídy, například validace dat nebo přihlášeného uživatele.
- **Converters** - obsahuje základní třídy pro konvertování jednotlivých vlastností, např. BoolTo-Visible converter.
- **VisualHelpers** - jsou zde umístěny pomocné třídy hlavně pro umožnění přepínání do tmavého režimu a zobrazování stromové struktury.
- **Components** - tato složka obsahuje vlastní komponenty, např. vlastní prvek na výběr data a času.
- **Resources** slouží na umístění ikonek a obrázků.

Obrázek 5.3: Přihlášení do aplikace

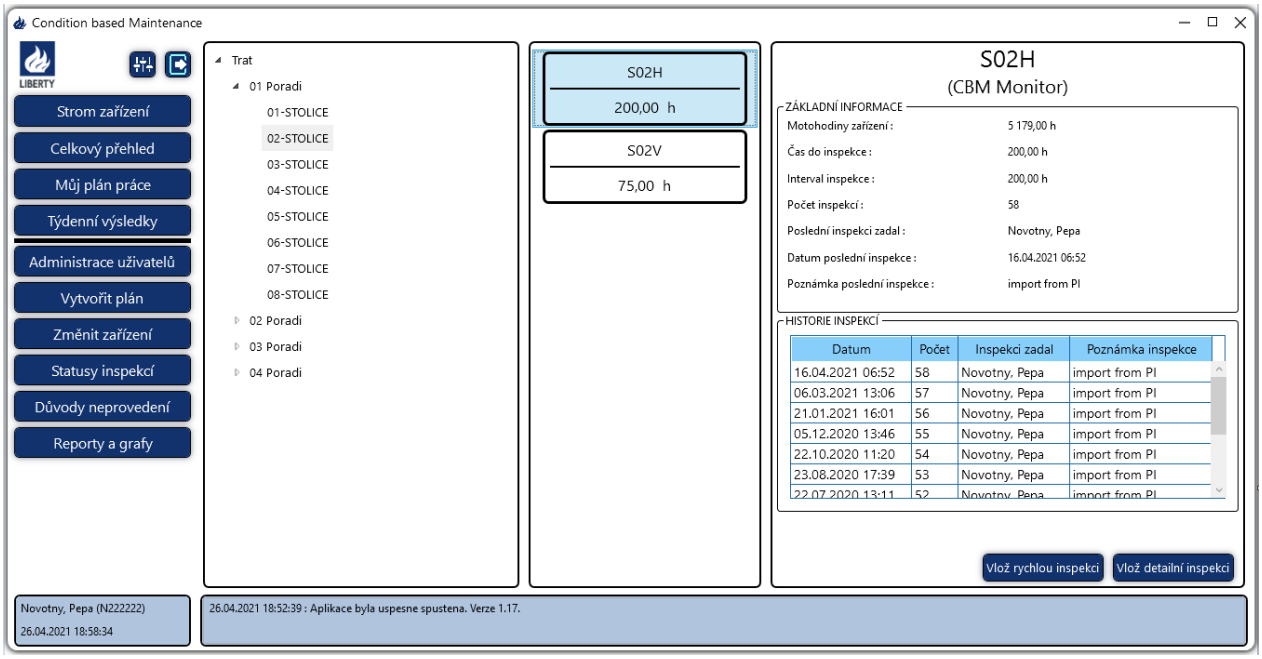
```
public static string ComputeHashFromText(string text)
{
   using (MD5 md5 = MD5.Create())
    {
      byte[] md5Data = md5.ComputeHash(Encoding.ASCII.GetBytes(text));
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < md5Data.Length; i++)
           sb.Append(md5Data[i].ToString("X2"));
        return sb.ToString();
    }
```

```
}
```

Listing 5.4: Metoda pro MD5 šifrování

Základním vstupem do aplikace je okno MainWindow.xaml, které spustí přihlašovací okno pro ověření uživatele. Hesla jsou v aplikaci šifrována pomocí algoritmu MD5.

Po ověření uživatele je dle jeho role nastaveno základní uživatelské rozhraní. Kde hlavní členění je menu s ovládacími tlačítky, dále na levé straně najdeme aktuálně přihlášeného uživatele včetně času. Napravo od menu poté najdeme hlavní část aplikace, kde se zobrazují jednotlivé komponenty právě zpracovávané akce. Ve spodní části aplikace je informativní panel, ve kterém uživatel nalezne 10 posledních oznámení systému. Toto bylo implementováno proto, aby nebyl uživatel "nucen" potvrzovat oznamovací okna, samozřejmě v případě chyby vyskočí okno s chybovou hláškou.



Obrázek 5.4: Uživatelské rozhraní aplikace

Jak již bylo zmíněno, klientská aplikace se stará pouze o zobrazení a sběr dat. Veškeré data připraví business vrstva, která je reprezentována aplikačním serverem. Tato komunikace funguje na principech RPC prostřednictvím WFC. Zde je uveden příklad, kdy klient chce získat seznam všech uživatelů daného oddělení.

```
internal List<TCbmUser> GetUsersByIdOperation(long? idOperation, bool
    includeDeleted)
{
```

```
        IWcfServices service = null;
        try
        {
           Connect();//připoj se k endpointu
           service = GetCommunicationChannel();
           return service.GetAllUserByIdOperation(idOperation,includeDeleted);
        }
        catch (Exception ex)
        {
            //logovani vyjimky
            return null;
        }
        finally { CloseChannel(service); }
    }
```

Listing 5.5: Metoda pro získání uživatelů daného oddělení

V neposlední řadě je třeba zmínit, že je zavedena jednoduchá kontrola uživatelských vstupů na základě implementace *IDataErrorInfo*. V jednotlivých XAML souborech pro jednotlivé okna se u dané kontrolky použije v bindingu vyvolání události o vzniku chyby. Ta je zpracována v base view modelu a informuje okno. Je nastaveno, že pokud se v okně vyskytují chyby, je blokováno použití tlačítka *Uložit*. Následuje krátká ukázku XAML kódu, který je použit při nastavení kontrolky reagující na chybu.

Obrázek 5.5: Ukázka chybějícího vstupu v dialogovém okně

```
<TextBox Grid.Row="0" Grid.Column="5" Style="{StaticResource
    CbmTextBoxError}" Text="{Binding EmplNumber, UpdateSourceTrigger=
    PropertyChanged, NotifyOnValidationError=True, ValidatesOnDataErrors=
    True, TargetNullValue=''}" />
```

Listing 5.6: Nastavení prvku na kontrolu vstupu

Hlavní výstupy z aplikace si lze opatřit ve dvou provedeních, pomineme-li zobrazení na obrazovce ve formě datagridu. Máme k dispozici vybraná data exportovat do tabulkového procesoru Excel, případně si nechat vygenerovat vizuální reprezentaci formou grafu, viz příloha B. Ty byly vytvořeny pomocí knihovny **Live Charts**, která je volně k dispozici a obsahuje velmi rozsáhlou podporu pro práci ve WPF. Na oficiálních stránkách [16] lze nalézt mnoho tutoriálů a příkladů k jednotlivých nastavením.

Export do Excelu probíhá ze stejného formuláře jako vytvoření grafu, tj. **Grafy a Reporty**. Funkce je implementována pomocí knihovny Closed XML [17]. Knihovna je k dispozici opět v licenci MIT (svobodná licence, která vznikla na Massachusettském technologickém institutu) a je velmi intuitivní k použití. Lze si tedy uložit *výpis plánovaných a provedených činností pracovníků*, dále také *detailní historii vybraného zařízení* a v neposlední řadě *10 nejčastěji opravovaných zařízení.*

Kapitola 6

Dosažené výsledky

Již během vývoje aplikace byly prováděny testy a konzultace s vedoucími pracovníky jednotlivých oddělení. Tímto postupem se minimalizoval počet úprav, které bylo nutné udělat po nasazení aplikace. Navíc byli uživatelé nadšeni, že jejich nápady se promítly do finální aplikace a tím jí lépe vzali za svou.

Jako zaváděcí oddělení byl vybrán závod Z14 SJV (středojemná válcovna). Po instalaci databáze, včetně navedení základních dat, instalaci aplikačního serveru a kontrole komunikace s PI Systémem, byla poskytnuta klientská aplikace 3 uživatelům. Jednomu vedoucímu a dvěma operátorům. Proběhlo krátké zaškolení, které obsahovalo hlavně přihlášení, kontroly plánů každého pracovníka a také vložení údržby. A to jednak tzv. rychlé, kde se nevybírají žádné možnosti, tak také detailní údržbu s možností výběru několika parametrů. Vedoucí poté dostal školení zvlášť, aby se dozvěděl jak obsluhovat uživatelská data, kontrolovat a měnit plány. V neposlední řadě bylo ukázáno, jakým způsobem dostat ze systému informace a reporty.

Testovací provoz trval celý týden. Během tohoto týdne byl odhalen problém při automatickém plánování údržby pro zařízení s intervalem údržby blízkým 24 hodin. Toto zařízení se nedostávalo do plánu nikdy, jelikož nebylo možné naplánovat údržbu den před vypršením limitu. Pro tento typ zařízení byl zvolen jiný způsob plánování, který při zjištění nutnosti zadání do plánu naplánuje údržbu na následující směnu. Další výtka byla směřována k nedostatečnému chybovému hlášení. Jinak řečeno, bylo požadováno detailnější rozepsání důvodů selhání akce, např. neuložení záznamu o údržbě. To je následek toho, že funkce vrací hodnoty bool jako výsledek. Aby se tomuto předešlo, bude nutné přepracovat návratovou hodnotu na objekt, který si ponese výsledek i informaci, proč provádění selhalo. Poslední připomínka byla k chybějící možnosti zadat různé typy údržby, kromě namazání také např. výměnu, očištění atd.

Jiné problémy nebyly pozorovány ani na datové, ani na serverové vrstvě. Obě vrstvy komunikovaly bez chyby, nedocházelo ke zvyšování počtu využitých vláken nebo nárůstu spotřeby paměti. Po celou dobu byly kontrolovány logy serverové části, kde nebyly nalezeny žádné provozní problémy.

Zadáno bylo celkem 102 záznamů o údržbě, kde pouze 2 z nich nebyly uloženy do databáze, a to z důvodu špatně zvolených vstupních údajů, tj. údržba byla vložena před datum poslední údržby.

Klientská aplikace byla přijata pozitivně. Byl velmi kvitován tmavý režim, dále také rychlost a jednoduchost aplikace. Reakční doba pro uložení dat do databáze byla dle zkušeností uživatelů velmi malá, ale to bude více ověřeno v ostrém provozu více uživatelů. Padly také návrhy na zlepšení a to:

- Zvětšit písmo v aplikaci.
- Rozšířit počet detailních parametrů, které se dají v rámci údržby zaznamenat - množství maziva, typ maziva atd.
- Detail údržbového místa by se mohl rozšířit o fotografie a základní popis postupu údržby, díky kterému by jednodušeji proběhlo zaškolení uživatelů.
- Vytvoření mobilní aplikace, minimálně ve verzi zadání údržby přímo na pracovišti.
- Reporty, které jsou pouze v aplikaci nebo v MS Excel podobě, mít možnost exportovat do PDF a přímo tisknout.
- SMS alarmy.
- Chybové hlášky rozšířit o detailní popis důvodu selhání akce.

Takže celkově vzato z ohlasů vyplynulo, že aplikace předčila očekávání. Sice nedosahovala možností a rozsahu, které byly nabídnuty specializovanými firmami, ale na druhou stranu byla aplikace "ušitá" na míru, v relativně krátkém časovém úseku a hlavně mnohonásobně levněji. Jako další benefit se považuje, že je aplikaci možno dále rozvíjet a upravovat.

Kapitola 7

Závěr

Cílem práce bylo vytvořit aplikaci pro podporu údržby na základě metodiky Cmb - Condition based Maintenace. Bylo tak rozhodnuto hlavně z důvodu, že nabízené hotové systémy byly drahé. A to hlavně z důvodu nutnosti koupit si celé řešení obsahující i ty části, které nejsou úplně potřeba. Je třeba říci, že daný cíl se podařilo splnit.

Zvolené technologie se ukázaly jako dostatečné a nebyl problém je využít k vývoji finálního programu. Rozdělení aplikace do tří vrstev bylo sice implementačně náročnější, avšak do budoucna dává prostor k jednoduššímu rozvoji. Nebude obtížné vytvořit další prezentační vrstvy jako mobilní nebo webovou. Další benefit této architektury je v tom, že pokud padne rozhodnutí změnit databázi, bude potřeba jen minimum úprav a klientské vrstvy se to nedotkne vůbec.

Z testování vyplynulo, že aplikace je uživatelsky přívětivá a hlavně byla oceňována rychlost aplikace. Zaškolování proběhlo bez problémů a již po prvním dni byli uživatelé schopni samostatně fungovat. Díky přihlašování jsou akce prováděny jmenovitě, a tím se podařilo odstranit částečnou anonymitu úkonů. Aplikace tak nabízí řadu možností, jak lépe plánovat a vést údržbu. Systém je navržen tak, aby šly přidávat na sobě nezávislé oddělení a tímto rozšířit možnosti nasazení aplikace.

Literatura

1. *Asset performance management* [online] [cit. 2021-03-29]. Dostupné z: https://www.appa.org/asset-performance/.
2. BENGTSSON, Marcus; OLSSON, Ella; FUNK, Peter. Technical Design of Condition Based Maintenance System. 2004. Dostupné také z: https://www.researchgate.net/publication/2877334_Technical_Design_of_Condition_Based_Maintenance_System.
3. *Maximo IBM* [online] [cit. 2021-03-29]. Dostupné z: https://www.ibm.com/products/maximo.
4. *COMOS SIEMENS* [online] [cit. 2021-03-29]. Dostupné z: https://new.siemens.com/global/en/products/automation/industry-software/plant-engineering-software-comos/portfolio.html.
5. *moneo ifm* [online] [cit. 2021-03-29]. Dostupné z: https://www.ifm.com/gb/en/shared/moneo.
6. *Třívrstvá architektura* [online] [cit. 2021-03-29]. Dostupné z: https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture.
7. ŠEC, David. *Distribuované systémy*. 2015. Dostupné také z: https://theses.cz/id/wow3pb/STAG84353.pdf. Diplomová práce. Univerzita Hradec Králové.
8. *ECMA-334*. Dostupné také z: https://www.ecma-international.org/publications-and-standards/standards/ecma-334/.
9. *C# language* [online] [cit. 2021-04-12]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.
10. *WPF Overview* [online] [cit. 2021-04-12]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/desktop/wpf/introduction-to-wpf?view=netframeworkdesktop-4.8.
11. PAŽOUREK, Tomáš. *Webové služby v architektuřeREST na platformě .NET*. 2012. Dostupné také z: https://is.muni.cz/th/u2ra1/thesis.pdf. Bakalářská práce. Masarykova univerzitaFakulta informatiky.
12. *MS SQL Server 2019* [online] [cit. 2021-04-12]. Dostupné z: https://docs.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-version-15?view=sql-server-ver15.

13. *Visualizing PI System Data*. 2020. Dostupné také z: http://cdn.osisoft.com/learningcontent/pdfs/VisualizingPISystemDataWorkbook.pdf.

14. FOWLER, Martin. *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley, 2003. ISBN 0-321-12742-0.

15. *Logging services* [online] [cit. 2021-04-12]. Dostupné z: https://logging.apache.org/log4net/.

16. *Live Charts* [online] [cit. 2021-04-12]. Dostupné z: https://lvcharts.net/.

17. *Closed XML* [online] [cit. 2021-04-12]. Dostupné z: https://www.nuget.org/packages/ClosedXML/.

Příloha A

Nasazení systému

Ke správnému fungování aplikace je zapotřebí vybudovat zázemí, tj. nainstalovat a zprovoznit databázi, nainstalovat webovou službu, která reprezentuje server a v neposlední řadě nastavit klienta. Dále je nezbytný zdroj provozních informací - PI Systém, kde lze načíst počáteční strukturu zařízení a jejich motohodiny. Pokud tento zdroj není k dispozici, lze aplikaci přepnout do testovacího režimu a využít pouze data uložené v databázi. Toto nastavení provedeme přepsáním položky enviroment v soubory *config.ini* z prod na test. Samozřejmě je třeba si uvědomit, že bez zdroje motohodin, nebude moct aplikace vyhodnocovat jak stav zařízení tak i plánovat jejich údržbu, pouze poskytne přehled nad daty.

Databáze byla zvolena Microsoft SQL Express 2019. K vytvoření databáze a testovacích dat je nutné se přihlásit pod uživatelem, který má právo zakládat databáze a tabulky, pod tímto uživatelem založit databázi, která bude sloužit pro uchovávání dat. Poté pomocí **SQL Server Management Studia** spustit skript *CmbTestData.sql*, který byl přiložen k vlastní aplikaci. V tomto skriptu jen změníme název vytvořené databáze. Databázi není nutno nijak speciálně nastavovat, je však třeba dohlédnout, aby byla správně nastavena pro přístup ze sítě (na firewallu serveru povolit port 1433).

Aplikační server, tj. servisu lze nainstalovat pomocí přiloženého souboru *CbM.IssServiceRegistration.bat* a to na libovolném serveru, který má přístup k databázi, tj. klidně i na stejný server jako databázi, i když doporučené to není. Po instalaci služby a před jejím spuštěním je nutno nastavit soubor *config.ini*, kde je třeba v položce **MES.Cbm.ISS.IssRepository** nastavit připojovací řetězec (a to pro klíč test i prod) ve tvaru

Data Source=adresa_databazoveho_serveru;Initial Catalog=CBM_Database;
User ID=jmeno_uzivatele;Password=heslo_uzivatele

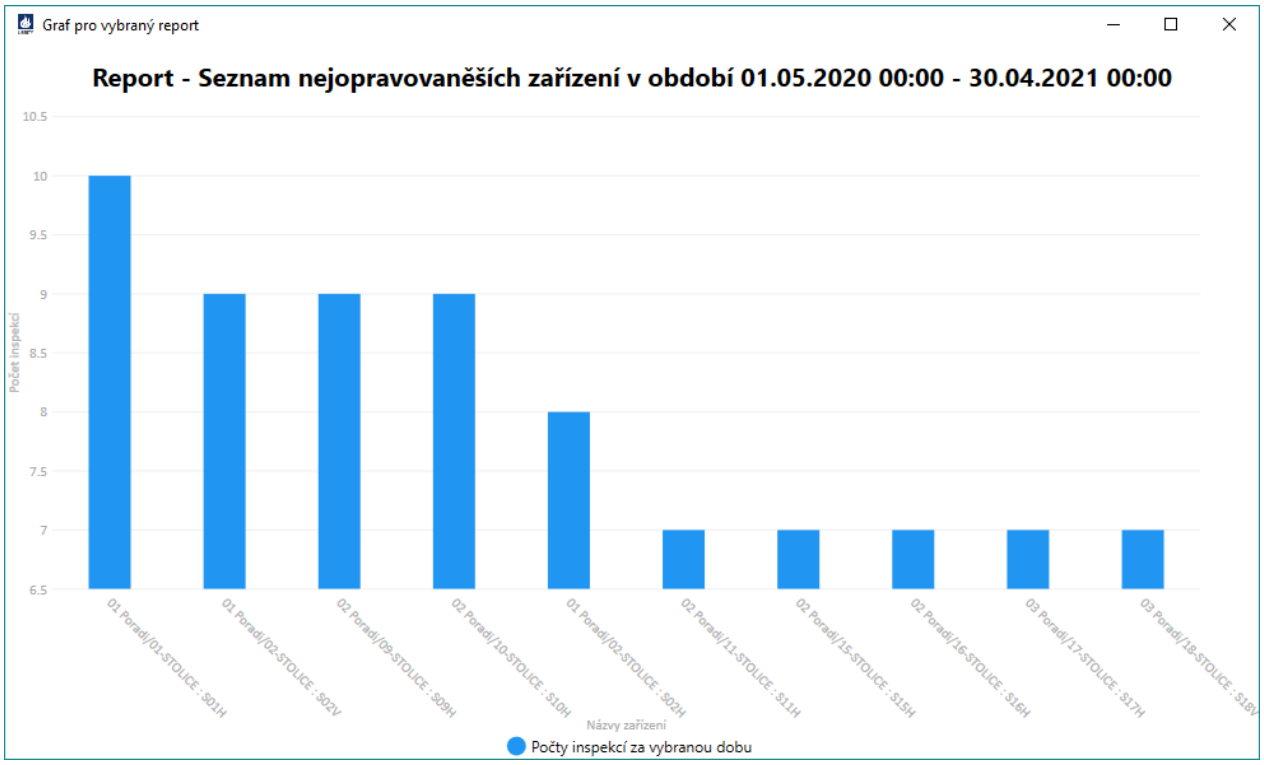
Poté již servisu spustit pomocí *Services.msc*. Je třeba počkat než se server inicializuje, co můžeme zkontrolovat v logu. Lze také využít toho, že aplikační server je možné spustit jako konzolovou aplikaci a tím pádem získat výpisy přímo do konzole. Tento způsob se doporučuje pouze na ladění, nikoliv na ostrý provoz. Soubor *MES.Cbm.ISS.exe* je nutné spouštět s právy administrátora.

Vlastní klientskou aplikaci není nutné instalovat, pouze změnit v souboru *MES.Cbm.WPF.exe.config*

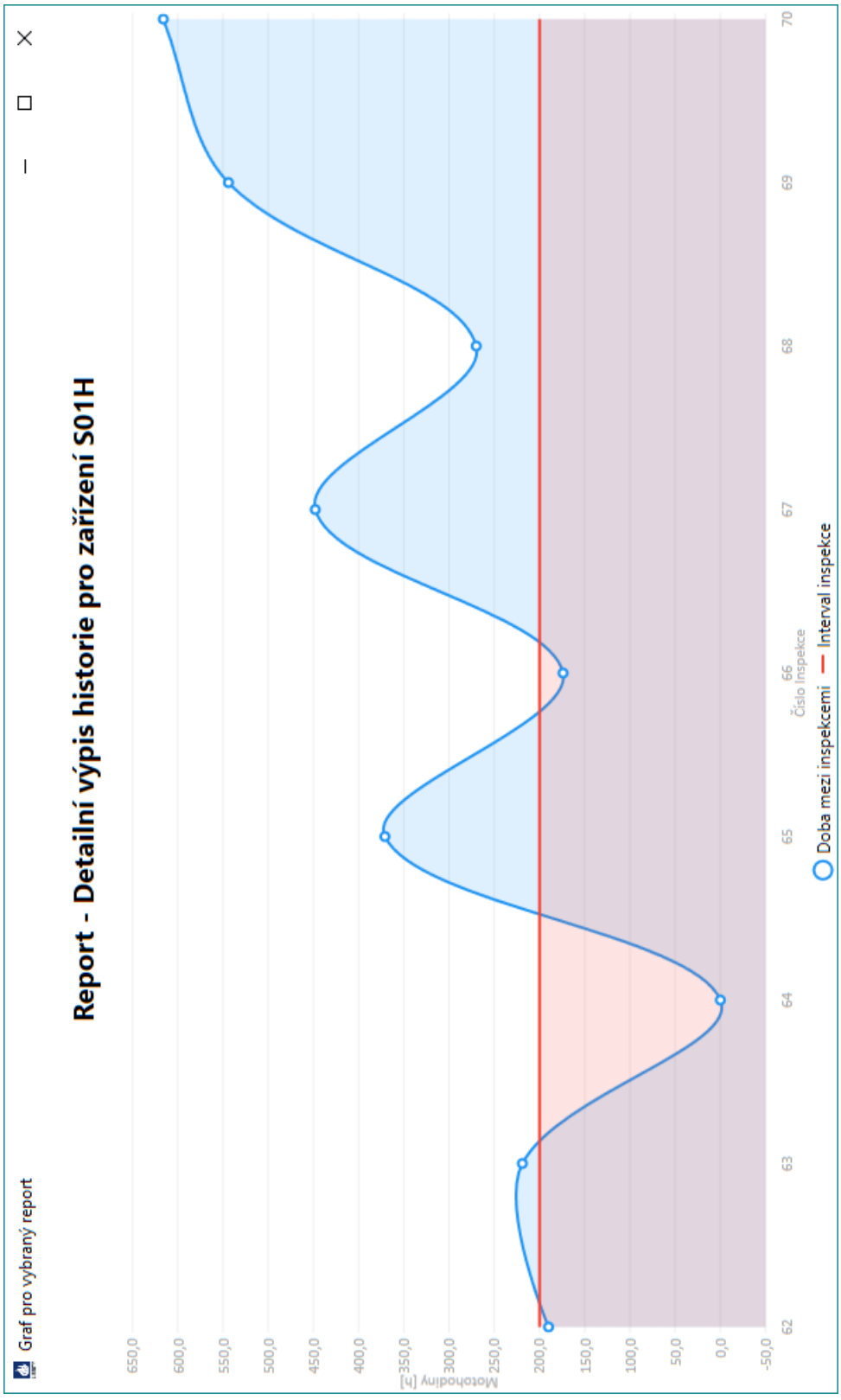
adresu k aplikačnímu serveru (položka s klíčem AppServer) : **http://ip_adresa_serveru:7100/Cbm**. K přihlášení použijeme předpřipravený účet s přihlašovacím jménem N222222 a heslem N222222.

Příloha B

Grafy



Obrázek B.1: Nejčastěji opravovaná zařízení



Obrázek B.2: Detailní historie zařízení